

AD-A188 198

A WORKED EXAMPLE OF AN APPLICATION OF THE SAINT  
SIMILATION PROGRAM(U) AERONAUTICAL RESEARCH LABS  
MELBOURNE (AUSTRALIA) S 551110 SEP 87 ARL-113-1M-23  
DODM-AR-004-354 P/C 12/3

1/1

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART

ARL-SYS-TM-93

AR-004-554



DTIC FILE COPY

DEPARTMENT OF DEFENCE  
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION  
AERONAUTICAL RESEARCH LABORATORIES

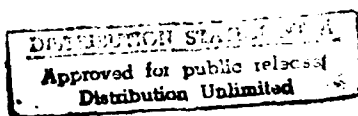
MELBOURNE, VICTORIA

Systems Technical Memorandum 93

A WORKED EXAMPLE OF AN APPLICATION OF THE  
SAINT SIMULATION PROGRAM (U)

by

S. SESTITO



Approved for public release.

This work is copyright. Apart from any fair dealing for the purpose of study, research, criticism or review, as permitted under the Copyright Act, no part may be reproduced by any process without written permission. Copyright is the responsibility of the Director Publishing and Marketing, AGPS. Inquiries should be directed to the Manager, AGPS Press, Australian Government Publishing Service, GPO Box 84, Canberra, ACT 2601.

SEPTEMBER 1987

AD-A188 198

DTIC  
ELECTE  
DEC 09 1987  
S D

AR-004-554

DEPARTMENT OF DEFENCE  
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION  
AERONAUTICAL RESEARCH LABORATORIES

Systems Technical Memorandum 93

A WORKED EXAMPLE OF AN APPLICATION OF THE  
SAINT SIMULATION PROGRAM (U)

by

S. SESTITO

SUMMARY

SAINT is a network modelling and simulation technique developed to assist in the design and analysis of complex human-machine systems. This document discusses some of the SAINT concepts, the development of a SAINT network, and the method of inputting the network into the SAINT environment, and presents a brief look at the output from SAINT.



(C) COMMONWEALTH OF AUSTRALIA 1987

---

POSTAL ADDRESS: Director, Aeronautical Research Laboratories,  
P.O. Box 4331, Melbourne, Victoria, 3001, Australia.

# CONTENTS

# PAGE NO

1. Introduction	1
2. Network	
2.1 Definition of a network	4
2.2 Development of an example network	4
3. The main features of SAINT	
3.1 Brief overview of SAINT features	5
3.2 Preliminary notes about SAINT's simulation program	5
3.3 Basic structure of networks	8
3.4 Flow of network	8
3.5 Time	9
3.6 Information attributes	9
3.7 Resources	10
3.8 System attributes	10
3.9 Access to attributes	11
3.10 User-functions	11
3.11 Moderator functions	12
3.12 Task statistics	14
3.13 User-generated statistics based on observations	14
4. Method of inputting the network into SAINT	
4.1 An overview of the input process	15
4.2 General information data	15
4.3 Inputting the task characteristics	16
4.4 The complete input needed for the example network	17
5. Running the SAINT system on the VAX computer at ARL	18
6. Output	20
7. Available manuals	22
Author's note and acknowledgements	23
References	24
Appendix 1.	25
Appendix 2.	27
Appendix 3.	28



Accession For	
NTIS CRASH	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Dist	Avail and/or Special
A-1	

## 1. INTRODUCTION

Many scientific and technical applications, including Human Factors applications, require the ability to represent a system in an abstract manner. In this way the influence of any modifications to the system can be simulated and the results analyzed. A simulation provides a powerful tool in enabling a system to be analyzed and improved. A system can usually be represented by a network, where a network is defined as an abstract representation of a system. In the case of a human-machine system, the network could include the various tasks to be performed, the time taken to perform these tasks, and the possible successive tasks. SAINT is a network modelling and simulation technique developed to assist in the design and analysis of complex human-machine systems. It was developed primarily by the Aerospace Medical Research Laboratory of the USAF for Human Factors applications in order to examine operator performance effects on system capability. SAINT is an acronym for Systems Analysis of Integrated Networks of Tasks. A network represented in SAINT concepts can be manipulated to test the effects of changing parameters, the task data or the branching logic. Therefore, given a network which depicts a particular system, the SAINT simulation program can be used to simulate this network and will give information about the behaviour of the system under various conditions which the user can specify.

There are basically two methods of representing the path(s) through a network. The first method is a simple path network which has only one possible path through it. Here, all the tasks are performed in a sequential manner which the designer or compiler orders in a pre-defined way. The second method is a multiple path network where the possible paths through the network are determined by the branching logic of each task. This branching logic can be probability-based, conditionally-based or sequentially-based. Therefore, in this type of network there are several paths through the network and these paths actually represent possible ways through the system. The SAINT technique provides the various types of branching logic in order to simulate real or proposed systems. In the following discussion, emphasis is on multiple path networks because the systems dealt with are too complex to be dealt with adequately, if at all, by single path networks.

The data for a simulation network can be derived from observations of real systems, experiment and laboratory part-task simulation, or even theory-based predictions. The nodes of the networks are actually meant to represent the performance of each task in terms of the time required for the successful completion of that particular task. Thus, in a network designed to approximate a proposed system, the data for the tasks can be generated by observations of real systems, laboratory part-task simulation involving human operators, or theory-based predictions. The network branching logic is also a function of observations from the real system or hypothesized network relationships. For example, the data could include the number of times that a recording function was performed by an operator. By comparing this value with the number of

times that the recording function was not performed, the designer of the network will be provided with the probability values of whether the recording function will be performed or not. Therefore, with a knowledge of the real working system and data representing the times required for the task completion and the interaction between the tasks, a network can be constructed.

The network in Figure 1 will be used as an example network to illustrate both the development of a network and the SAINT concepts needed to implement it as a simulation. The tasks in this network are fully explained in the following sections.

The SAINT technique provides a simulation computer program (which has been called SAINT.FOR) which follows the characteristics of the network. This program, written in Fortran, accepts networks specified in SAINT concepts; these concepts include task descriptions, branching logic and task duration. By understanding the SAINT concepts, one can therefore model a network according to these concepts and thus input this network into the simulation program.

Some of the basic SAINT concepts are discussed in the following text. The method of inputting this network into a SAINT format is also important and this is discussed below as well.

This document is intended to provide a novice with enough information to construct a network, input the network into the SAINT simulation program and obtain meaningful results pertaining to the system. The example system in Figure 1 is used to demonstrate the development of a network, the SAINT concepts used to implement the network and the method of inputting this network into the simulation program. The output from this example system is discussed also.

This document outlines some of the lessons learnt from attempting to use the SAINT computer package which was provided through The Technical Cooperation Program (TTCP) Subgroup U from the USA to the Human Factors Group at ARL.

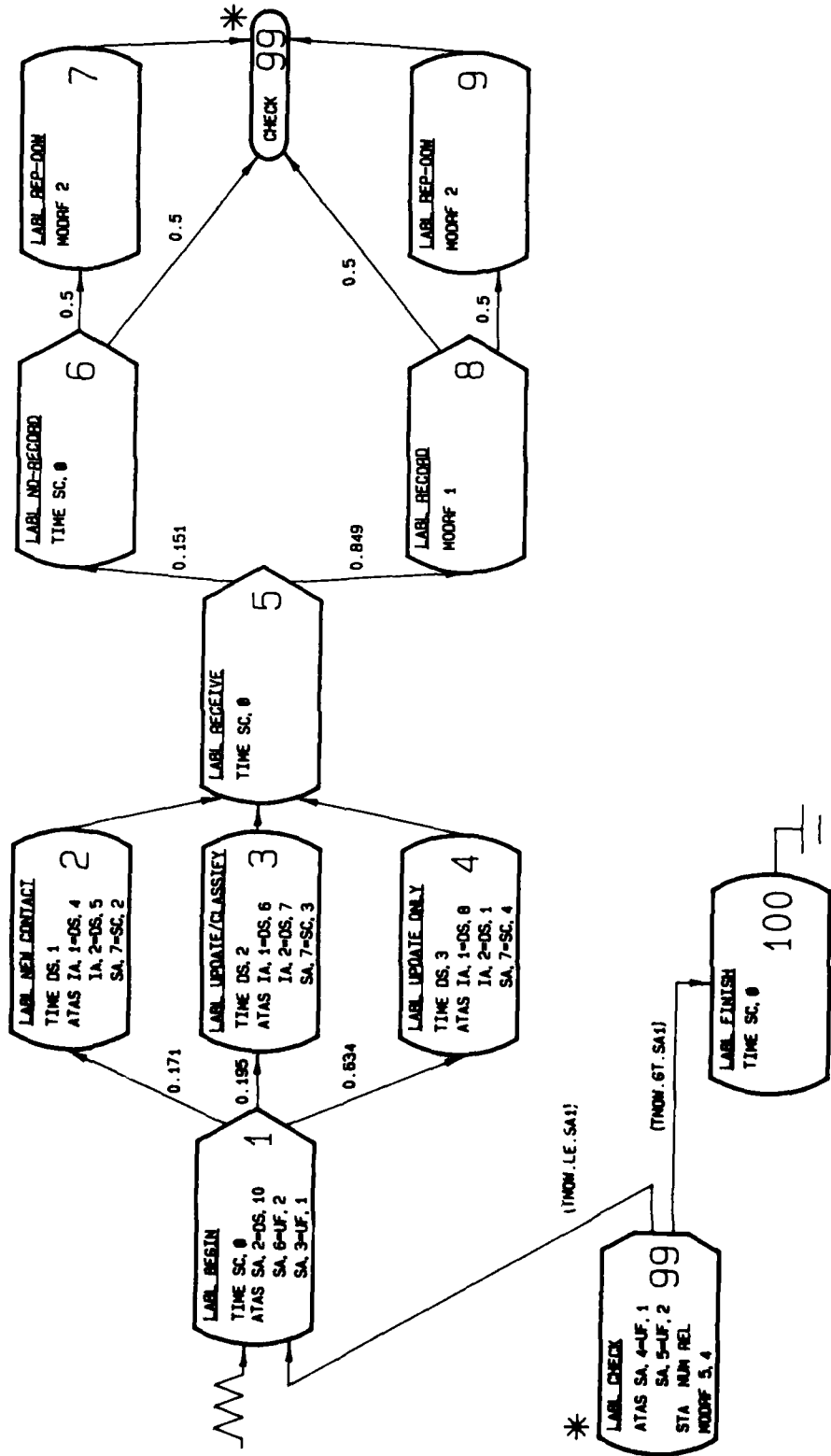


Figure 1: Example network of an operator processing incoming messages.

## 2. NETWORKS

### 2.1 Definition of a network

A network in the present context is an abstract representation of a model of a real working system. It includes a description of the various tasks and the various links between the tasks.

### 2.2 Development of an example network

A network can be developed from a knowledge of a real system and data which provide the necessary numerical values of the system.

As a suitable example for the demonstration of the development of a network, consider a system which simulates a shipboard console operator processing incoming messages. The development, inputting and testing of the network representing this system is discussed in the following text. The developed network is shown in Figure 1. Some of the main features of this network are

1. there are three types of incoming messages,
2. the rate of message generation is varied between two limits and is controlled by the SAINT controlling program, and
3. the whole scenario is run for a fixed period of time (time limit).

These features were included on the basis of actual shipboard observations. Limits on rate of message generation and the run time limit have been set by human limitations and practical considerations, including those of economical use of computer time. The two activities the operator can perform upon the messages are

1. RECORD : recording information, and
2. REP-OOW : reporting to the officer of the watch (OOW).

These two activities represent the only tasks in the system which can be applied to the incoming messages by the operator.

In order to collect data from the system simulation a CHECK task is inserted into the network. The CHECK task does not influence the function of the operator but it collects data on the network operations. The CHECK task has the following function. The arrival time of a message is determined by the SAINT program which uses a distribution set entered by the user. Thus the time at which each message is generated will be different for each message. From these specifications two cases can occur

1. if the next message generation begins after the previous message has been completely dealt with, the operator must 'wait' for the next message, or
2. if the next message generation begins before the previous message has been completely dealt with, this message must be placed on to a queue and processed as soon as the operator is free.

Thus the CHECK task will either wait or not for the next message and this activity will be controlled and monitored by the CHECK task itself. The task duration of CHECK will therefore not be constant. CHECK is implemented with the use of a moderator function (which will be explained below).

The experimental data used to construct the network shown in Figure 1 included approximately 200 messages. These messages were analyzed in the following manner. The time taken to perform the various activities, i.e. recording information and reporting to the OOW, was recorded for each message. For example, if a message did involve recording information then the variable corresponding to recording information (RECT) was filled with the associated recording time; on the other hand if no recording was performed then this variable remained set to zero. Each message analyzed in this way thus represents a path through the system. As an example, if a message contained non-zero values for RECT and reporting to the officer of the watch (REP-OOW) then this indicates that both the recording and reporting to the OOW activities were performed; so one path through the system involves doing both the recording and reporting to the OOW activities consecutively. Thus the different paths through the network represent the different types of incoming messages.

Through a knowledge of the system the designer of the network can order the activities; for example, whether the recording information activity was performed before or after the activity of reporting to the OOW.

The possible paths through the system are fixed by the branching routine determined after a task has been completed. The branching types are determined by the experimental data. If the designer of the network is at a certain point in the network (e.g. just after the completion of the RECORD task) then by considering the next possible activity the probability value for that activity to be performed can be determined. For example, by counting the number of messages which have non-zero values for REP-OOW (which satisfy the pre-requisites of RECT > 0) then the probability of performing this activity can be obtained; this probability would be equal to the number of messages which have non-zero quantities for REP-OOW divided by the total number of messages being considered at this point.

Since most of the nodes (junctions) throughout the network were determined in this manner the corresponding tasks have branches based on probability values. Some of the tasks are only buffer tasks (that is, take no time to perform) which aid in the design of the network in

utilizing these determined probability values; an example of this is the BEGIN task. Some of the tasks in the network are also duplicated (i.e. perform the same function) in order to clarify and utilize these determined probability values; an example of this is the REP-OOW task. Most of the other branches are deterministic; that is, all the tasks emanating from this task will be performed. The CHECK task has the only conditional branching. Here, the current elapsed time is compared to the time limit specified by the user. If the elapsed time is greater than the time limit, then the simulation performs the FINISH task and then stops; if this is not the case then the system starts again at the BEGIN task and a new message is generated. This CHECK task, as mentioned before, also controls the waiting or non-waiting period of the operator in connection with the arrival of the next message.

The network shown in Figure 1 will be used to illustrate some of the SAINT concepts. The method of inputting this network into the SAINT simulation program will be then demonstrated. In particular, the implementation of the key CHECK task will be discussed.

### 3. MAIN FEATURES OF SAINT

#### 3.1 Brief overview of SAINT features

The following sections contain a brief explanation of some of the SAINT concepts. Many of the concepts are illustrated with the use of the example network found in Figure 1. The purpose of this section is to provide enough knowledge for a newcomer to construct and develop a simple SAINT network. This section complements the detailed description of all of the SAINT concepts given in the SAINT manual called 'Simulation using SAINT: a user-orientated instruction manual' (Ref. 1).

#### 3.2 Preliminary notes about SAINT's simulation program

The SAINT computer simulation program (which has been called SAINT.FOR), is written in the FORTRAN 4 or FORTRAN 66 programming language. There were four modifications made to SAINT.FOR as received from the US to make it compatible with the Fortran available on the ARL VAX-11/780 computer. Firstly, line 11880 was missing (the line numbers are found at the end of each line); the missing code was obvious and easy to replace. Secondly, the function DRAND is a machine dependant function. DRAND generates a random number and it was modified to be compatible with the VAX computer. Thirdly, since the files 18 (nrnit) and 19 (nrent) are only used in the program, it was decided to make them internal only; that is they were deleted at the end of the program run. Finally, the format statement of line 8400 was modified to allow the output printing of a larger variable; in Fortran, output is printed according to the specifications found in a format statement and with the example network it was found that the specifications of a particular variable allowed insufficient space for its proper output printing. A listing of the modifications made to the program can be found in Appendix 1.

The exchange of variables and values between the various parts of the simulation program are carried out in COMMON blocks, which are found at the top of various sub-programs (just under each header). To access a variable, the common block which contains this variable must be included in the sub-program. Only those common blocks which contain the variables referenced in the sub-program need to be included. For example, referring to Figure 2, because the variable TNOW is used in this function only common block com06 is included here.

---

```

FUNCTION USERF(JJ)                                00050920
COMMON /COM06/ TNOW,TINEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
*           NRNIT,NRENT,MNDC,NDC,NDTN,NNTC

C      CODE FOR USERFUNCTION 1
10     IF JJ=1 THEN USERF=TNOW
      RETURN
      END

```

---

Figure 2. Example of the usage of the COMMON block.

---

The SAINT simulation program is made up of many subroutines and functions. Most of these sub-programs are completely coded and are used solely to accept, interpret and simulate a network. The SAINT.FOR program also provides sub-programs which can be changed by the user in order to reflect certain activities pertaining to a particular network. Two examples of these adjustable sub-programs, Function Userf and Subroutine Modrf, are discussed below.

### 3.3 Basic structure of networks

The basic element of a network is a task; a task represents a particular function that is to be performed. All of the tasks in Figure 1 are acceptable task symbols. Each task has a label and a number. Task description codes are used to define the specifications of a task and they are found on the left-hand side of the task symbol. The task description code for the label of the task is 'LABEL'. The number of the task is found on the right-hand side of the task symbol. There are three types of tasks available in the SAINT system; source tasks, ordinary tasks and sink tasks. A task which starts at the beginning of a simulation is a source task; it has no preceding task at the beginning of a simulation. The BEGIN task in Figure 1 is an example of a source task. An ordinary task is a task whose sole purpose is to perform its function and to indicate by the branching logic the possible succeeding tasks; an example of this is the RECORD task in Figure 1. A sink task is a task whose completion could cause the stopping of the simulation; the FINISH task in Figure 1 is an example of a sink task.

### 3.4 Flow of the network

The flow of any network is determined by the branching logic of each task. There are four types of branches available in the SAINT system.

Deterministic branching (D) indicates that all branches emanating from this task will be selected; that is, the probability of selecting the emanating tasks is 1.0 for each task. The successive task that is selected from a task with probabilistic branching (P) is determined solely by probability values that are placed on each branch. In this

case, only one branch is selected and this branch is selected on probability. There are two types of conditional branching that can be specified. Each of the tasks emanating from these types of branching have conditions placed on them. For the conditional-take first branch ( $\square$ ), the first branch where the conditions are satisfied is selected. For the other type, the conditional-take all branch ( $\square$ ), all the branches whose conditions are satisfied are selected.

### 3.5 Time

The SAINT.FOR program simulates a network in a time-dependent manner. For each task, the amount of time required to perform its function has to be specified. The task description code for task duration is 'TIME'. There are basically three methods of specifying the task duration. The first method is a simple method in which the time required is a constant value; this value is represented as a scaled constant (SC). The second method is where the task duration can be selected from probability distributions which are provided by the user in items called distribution sets (DSs). The values for the DSs are derived externally to the program and are based on the knowledge of the system activities. The SAINT simulation program selects a value from this distribution and assigns this value to the task duration. In this way the network has a better face validity as the time to perform a particular task will not be constant. The third method requires the use of user-functions (UF) and these will be discussed later.

It is important to note that values from the DSs can be used in any assignment statement. That is, assigning a value from a probability distribution is not confined to the assignment of task duration.

### 3.6 Information attributes

Information is passed through the network, that is, along the various branches, by information packets which contain various attributes. At the start of the simulation, information packets are created for all source tasks. The task description code for ATtribute ASSignment is 'ATAS' and this code followed by the information attribute specification IA, indicates that an IA assignment is to be performed. Assignment of these attributes can be made at any task in the network.

As an example of the usefulness of IA, refer to the example network in Figure 1. At the tasks 2, 3 and 4, assignment of two IAs is performed.

At task 2 (NEW CONTACT messages),  
     IA 1 = value from distribution set (DS) 4 and  
     IA 2 = value from DS 5.  
 At task 3 (UPDATE/CLASSIFY messages),  
     IA 1 = value from DS 6 and  
     IA 2 = value from DS 7.  
 Similarly for task 4 (UPDATE ONLY messages),

IA 1 = value from DS 8 and  
 IA 2 = value from DS 9.

For clarification, IA 1 contains the value of recording time (RECT) and IA 2 contains the value of the time taken to report to the officer of the watch (REP-OW). The purpose of this is so that, if, for example, the RECORD task is to be performed, the value of RECT will be obtained from the value of IA 1. In other words, the three different types of messages, described as tasks 2,3 and 4, have different values for RECT and REP-OW and so, depending on the type of message being generated (determined by tasks 2,3 or 4), the values of RECT and REP-OW will be selected. This is important because, as the example network is simulated for a time limit set by the user, different types of messages will be generated and the differing values of RECT and REP-OW will be required.

### 3.7 Resources

Resources are defined as any non-consumable commodities that are required for the performance of one or more tasks. A trivial example of a resource would be the availability of an operator to process the incoming messages. Further information on resources can be obtained from the SAINT manuals which are listed at the end of this report. A brief description of these manuals is found in Section 7. The information on resources is found in Ref. 1.

### 3.8 System Attributes

System attributes (SAs) are pieces of information which do not flow through the system, but are global in nature in the sense that they are accessible to all tasks. There is only one set of SAs associated with a SAINT model. Assignment to SAs can be made at any task by once again using the task description code of 'ATAS'; here this code is followed by an SA which indicates an SA assignment.

As an example of the usefulness of SAs, refer once again to the example network in Figure 1.

At task 1, BEGIN,

SA 3 = value from user-function 1.

These user-functions (UFs) are discussed in section 3.10. Essentially, SA 3 is assigned the value of the current elapsed time.

At task 99, CHECK,

SA 4 = value from UF 1 and

SA 5 = value from UF 2,

where UF 2 determines the difference between SA 3 and SA 4.

Essentially SA 5 will contain the time it took to travel along one path or branch. This information is important for the network depicted in Figure 1; it actually determines the task duration of the CHECK task.

### 3.9 Access to attributes

Access to attribute values, such as information or system attributes, can only be made through user-written sub-program calls to already-coded subroutines. User-written sub-programs, as indicated before, are adjustable sub-programs available to the user for the application of a particular network. As an example of accessing attributes consider the following: the SUBROUTINE GETSA(1,VALUE), a predefined subroutine, will obtain the value of SA 1 and assign it to the variable VALUE. A call to this pre-defined subroutine can be made at any part of the program, but it is only accessible to the user through user-written sub-programs. Two examples of user-written sub-programs are discussed in the following two sections.

### 3.10 User-functions

User-functions (UFs) are examples of the user-written subprograms available to the user. UFs enable us to write Fortran functions (under FUNCTION USERF) for generating attribute assignment values. At any task in the network, we can specify any attribute assignment (with ATAS) using the function specification UF. Internally, this directs SAINT to call function USERF with the UF number as an argument. The required attribute values are computed in the function USERF. SAINT automatically assigns these values to the attributes specified. In this manner, attribute assignment values can be computed as a function of any SAINT or user-defined variable. An example of the code for function USERF is shown in Figure 3.

---

```

FUNCTION USERF(JJ)                                00050920
COMMON /COM06/ TNOW,TINEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
*            NRNIT,NRENT,MNDC,NDC,NDTN,NNTC

C      CODE FOR USER-FUNCTION 1
10     USERF=TNOW
        RETURN

END                                            00050960

```

---

Figure 3. Example of FUNCTION USERF.

### 3.11 Moderator functions

Moderator functions are another example of the user-written sub-programs facility which is available to the user; they provide a method of changing the task duration by considering any SAINT or user-defined variable. The task description code for moderator functions is 'MODRF'. The program code for these functions must be written in Fortran under the SUBROUTINE MODRF(mfn,nnode).

Referring once again to the example network, this ability is needed to determine the task duration of the CHECK task. Remembering that the task duration of the CHECK task depends upon the time taken to traverse one path and the rate of message generation, the necessary code for this action is found in Figure 4. Also recall here that messages may be placed on to a waiting queue and this is the reason for using the CURLAG (current lag) variable in the subroutine in Figure 4. CURLAG will be either equal to the lag-time of the preceding message (i.e. the over time needed to deal completely with the message) or it will be equal to zero. The variable TTIME is the variable name of the task duration, and by assigning the determined value to this variable the task duration time is modified. Note also that the CHECK task also checks for the end of the simulation, i.e. the time limit, and if current time is greater than the time limit then the task duration of the CHECK task is zero; this is done to handle the case where the time limit has been passed and the task duration would be made equal to the difference of rate and traversal time. The call to the subroutine UCLCT in this subroutine is explained below.

All moderator functions are by default assumed to be deactivated; to apply them to a task, they need to be activated for the duration of that task only.

---

```

SUBROUTINE MODRF(MFN,NNODE)                                00031070
COMMON /COM22/ TTIME,PFIRB
COMMON /COM06/ TNOW,TINEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
*               NRNIT,NRENT,MNDC,NDC,NDTN,NNTC

C      CODE FOR MODERATOR FUNCTION 4
C      THIS MODERATOR FUNCTION DETERMINES THE TASK PERFORMANCE TIME
C      OF THE 'CHECK' TASK. THIS IS DEPENDENT UPON THE TIME THAT IT
C      TOOK TO TRAVERSE ONE BRANCH AND THE FREQUENCY TIME OF THE
C      MESSAGES
10     CALL GETSA(2,FREQT)
        CALL GETSA(5,BRANCHT)
        CALL GETSA(6,BRANLAST)
        CALL GETSA(7,TASKNO)
        ITASKNO=INT(TASKNO)

        CALL UCLCT(FREQT,17)

        CURLAG=FREQT-BRANCHT+CURLAG

        IF (CURLAG.GT.0) THEN
            TTIME=CURLAG
            CALL UCLCT(TTIME,13)
            CURLAG=0
        ELSE
            CALL UCLCT(-CURLAG,14)
            CALL UCLCT(BRANCHT,11)
            CALL UCLCT(BRANCHT,ITASKNO+3)
            TTIME=0
            IF (FREQT-BRANCHT.LT.0) THEN
                CALL UCLCT(BRANCHT,15)
            END IF
        END IF

C      CHECKING TO SEE WHETHER ELAPSED TIME IS GREATER THAN TIME LIMIT
        CALL GETSA(1,TIMELIMIT)
        IF (TNOW.GT.TIMELIMIT) TTIME=0

        RETURN                                              00031090

END                                                         00031100

```

---

Figure 4 : Example of the SUBROUTINE MODRF, with the necessary code for the CHECK task.

### 3.12 Task statistics

The SAINT system allows us to define task statistics for any task; SAINT obtains estimates of the mean, standard deviation, minimum and maximum associated with the statistical quantity to be observed. Two typical examples of task statistics would be the number of times that a task was completed and the number of occurrences of the releases of a task; note that a task may be released but may be unable to start because of the unavailability of resources. A useful statistic is the interval statistic. This statistic allows us to determine the time taken between two specified tasks. For example, to obtain the time between the release of task 1 to the completion of task 5, we would mark the release of task 1 and collect an interval statistic at the completion of task 5.

Therefore, task statistics provide the means for collecting statistical values for the various occurrences and interactions between tasks.

### 3.13 User-generated statistics based on observations

In a network, there can sometimes occur a situation where the statistical values wanted can not be obtained by collecting task statistics. For example, referring to the network in Figure 1, we want to know the mean traversal time of the messages that cause the succeeding messages to be put on to the waiting queue. The method of doing this is by using the SUBROUTINE UCLCT(x,y); this subroutine causes the value of x to be regarded as an observation of the user-generated statistic based on observations (UBO - User generated statistic Based on Observations) number y. Note that all UBOs must be defined at the beginning of the simulation. So, each time a value is put into this call, it is regarded as another observation. Thus the mean, standard deviation, minimum, maximum and number of observations can be calculated. Once again, this system-defined subroutine can only be accessed from a user-written subprogram. For the example network this is done through moderator functions 4 and 5 which put different values into different UBOs. To print and display the UBO values, the call, CALL SUBROUTINE UCLCT(XX,0), must be used. This will cause all the UBOs to be displayed. Note that the zero value is the important value in this call; the value of the real parameter XX is not used.

#### 4 METHOD OF INPUTTING A NETWORK INTO SAINT

##### 4.1 An overview of the input process

The SAINT input process works in the following way. All the information pertaining to a network is broken up into a series of input lines; each line is further subdivided into fields which are separated by commas. The first field of these input lines identifies the line and type of information that should follow. In the SAINT simulation program there are subroutines and functions which pertain to each possible input line; these subprograms interpret the entered information accordingly. For example, as will be further explained below, a data line with the first field equal to 'GEN' will cause SAINT to call subroutine GEN to interpret the values on this line. In the SAINT manuals (specifically Ref. 2), these data lines are referred to as cards and so in the following discussion the terms, input lines and cards, will be used inter-changeably.

All the input lines pertaining to a network must be created under the file 'FOR015.DAT'. The output is written to the file 'FOR016.DAT'. These two files are the only external files that need to be considered.

##### 4.2 General information cards

Before the task descriptions are put into the data input file, several general information lines should be included at the top of the data list. These lines provide information about the whole simulation system. As with all the input data cards, each field of a particular line has a particular meaning. As mentioned above, the first field identifies the card.

The basic initial lines that all SAINT simulation networks require include the GEN (general information), POP (program options) and OUT (output options) cards. The GEN card contains general information and includes the date and number of iterations (runs) required. The POP contains the program option details and includes the number of information attributes, system attributes and moderator functions being used in the simulation. Information about output options, such as whether a detailed iteration output is wanted and the iteration number for which the statistics task summary output should begin and end is all included on the OUT card.

If a particular field on a card is omitted (which is indicated by a comma or a blank and a comma or skipping to another field by indicating the field number in rounded brackets) then the default value for the field is assumed. Also, note that the termination of a card of information is indicated by an asterisk. For example, if the following card is entered

```
GEN,SABRINA,,,1986,(10)999*
```

these assumptions will be made

field 1 = GEN  
 field 2 = SABRINA  
 field 3 = default value  
 field 4 = default value  
 field 5 = 1986  
 fields 6,7,8 & 9 = default values  
 field 10 = 999  
 field 11 = default value.

The GEN card actually has 11 fields to be defined and even though the card is terminated at field 10, field 11 is still assumed to have a default value. The important question here is how it is known that the GEN card has 11 fields and what values are used as the default values. All of this information can be obtained from the SAINT User's Manual (Ref. 2). This manual has a list of all the possible input cards (on pages 39-85, Ref. 2), with the meaning of each field and its corresponding default value. This is necessary since all the fields need to have an appropriate value, either by definition or by default. For example, once again referring to the above GEN card, the default value that will be used for field 3 of this GEN card will be 1, as can be seen on page 39 of the SAINT User's Manual.

Other general information cards can be included at the top of the data input cards list, and these can include distribution set definitions and initial values for system attributes.

At the end of the input data lines, a 'FIN' line must be included to indicate that there are no more data lines to be read.

#### 4.3 Inputting the task characteristics

Information pertaining to a particular task can be contained on up to six lines. These lines (which are described on pages 59-71 of the SAINT User's Manual) are all related to the same task by field 2 on each of the cards; this field contains the task number.

The first essential card is a TAS card which contains information such as the type of task (i.e. source, ordinary or sink task) and the task duration. The following cards pertaining to tasks are only included if they will be used. The STA card is used if there are any marking or statistics to be gathered at this task. Similarly, an ATA card is used only if attribute assignments are to be made. A MOD card is used if a moderator function is to be applied to this task. A branching card is also only used if branching is to be performed; a DET card is used for deterministic branching, a PRO card for probabilistic branching, a CFI card for conditional-take first branching and a CAL card for conditional-take all. The definition of each field of the cards is in the SAINT User's Manual.

As an example of a task, refer to the CHECK task of Figure 1. This ordinary task (i.e. not a source or sink task), called CHECK and numbered 99 performs the following activities

1. obtains the values of system attributes 4 and 5; where SA 4 is assigned the value of user-function (UF) 1, and SA 5 is assigned the value of UF 2 ,
2. keeps count of the number of times that this task is released,
3. applies moderator function 5 in order to determine the values of the user-generated statistics,
4. determines the task duration from moderator function 4, and
5. branches this task to task 100 if current elapsed time is greater than the value specified in SA 1 (i.e. time limit) or to task 1, otherwise.

The input lines needed to describe this task would be as follows

```
TAS,99,CHECK,1,1*
ATA,99,REL,SA,,4,UF,1,SA,,5,UF,2*
STA,99,,,NUM,REL*
MOD,99,5,A,T,4,A,T*
CFI,99,1,TLA,1,,SA,,100,TGA,1,,SA* .
```

#### 4.4 The complete input needed for the example network

All the input needed for SAINT to simulate the example network is listed in Appendix 2; this listing would actually be created under the file 'FOR015.DAT'. Note that the meaning of each card can be found on pages 39-71 of the SAINT User's Manual (Ref. 2). The associated code for the user-written subprograms of SUBROUTINE MODRF and FUNCTION USERF can be found in Appendix 3.

## 5. RUNNING THE SAINT SYSTEM ON THE VAX COMPUTER AT ARL

In order to run the SAINT simulation program on the VAX-11/780 computer at ARL a complete copy of the simulation program SAINT.FOR is required. This program is available to qualified requestors through the USAF. (Note that Micro-Saint is available for IBM PC's and compatibles, but to date, the author has not used this package and thus can not make any comment on its capabilities.)

After creating the input file of 'FOR015.DAT', the procedure for running it is like that for any Fortran program; that is, the following sequence of commands is necessary

```
FOR SAINT<cr>
LINK SAINT<cr>
RUN SAINT<cr>
```

where <cr> = carriage return.

Because the SAINT simulation program is very long (approx. 7770 lines) the FOR command, i.e. the compilation, takes a relatively long time.

If the user-written sub-programs are to be utilized then the following procedure is definitely preferable and less time-consuming. The code for the user-written sub-programs should be moved from the program SAINT.FOR and placed into another file the author has called SAINTSUB.FOR. Next, the command

```
FOR SAINT<cr>
```

should be executed. This command will create the object file for the SAINT program without the user-written subprograms. This command will not have to be used again. Next the file with the subprograms should be compiled with the command

```
FOR SAINTSUB<cr>.
```

Now, to connect the two files the following command is necessary

```
LINK SAINT,SAINTSUB<cr>.
```

Finally to run the now-combined program the command

```
RUN SAINT<cr>
```

can be used.

If the user-written subprograms do not work as intended, then only the file Saintsub.for needs to be modified and compiled again with the command

```
FOR SAINTSUB.
```

The LINK and RUN commands are the same as above.

This procedure greatly decreases the compilation time (i.e. using the FOR command) of the program since only the parts that are changed, i.e. the user-written subprograms, are compiled again.

## 6. OUTPUT

The output created on the file FOR016.DAT contains all the information specified by the user as input. It partitions the input into useful blocks of information and thus allows a useful check of the input. It specifies the characteristics of the tasks, the initial values of the system attributes, the initial status of the moderator functions and groups together the tasks with the same branching type. The tasks at which attribute assignments are made are also listed (see Figure 5). In other words, the input is interpreted and displayed by the SAINT system in the interest of allowing the user to recheck the input.

*ATTRIBUTE ASSIGNMENT INFORMATION*						
TASK NUMBER	ASSIGNMENT POINT	ASSIGNMENT TYPE	RESR NUMBER	TRIB NUMBER	FUNCTION TYPE	PARAMETER SPEC
1	REL	SA		2	DS	10
		SA		6	UF	2
		SA		3	UF	1
2	REL	IA		1	DS	4
		IA		2	DS	5
		SA		7	SC	2
3	REL	IA		1	DS	6
		IA		2	DS	7
		SA		7	SC	3
4	REL	IA		1	DS	8
		IA		2	DS	9
		SA		7	SC	4
99	REL	SA		4	UF	1
		SA		5	UF	2

Figure 5 : Example of the output related to attribute assignment as given in the output file, FOR016.DAT.

The output requested with the OUT card is displayed at the end of the entered information. For example, a detailed iteration progress can be displayed or a summary of the task statistics and user-generated statistics can be shown. Figure 6 contains the requested user-generated output obtained from the example network.

**USER-GENERATED STATISTICS FOR VARIABLES BASED ON OBSERVATION**							
	MEAN	STD DEV	SD OF MEAN	CV	MINIMUM	MAXIMUM	OBS
NEWCON	0.1972E+02	0.2392E+02	0.1692E+02	0.1213E+01	0.2799E+01	0.3663E+02	2
UPCLASS	0.1055E+03	0.1081E+02	0.7647E+01	0.1025E+00	0.9787E+02	0.1132E+03	2
UPONLY	0.3025E+02	0.4813E+01	0.2779E+01	0.1258E+00	0.3440E+02	0.4365E+02	3
----							
	NOV VALUES RECORDED						
NEW CONQU							
	NO VALUES RECORDED						
UPCLQUE	0.1055E+03	0.1081E+02	0.7647E+01	0.1025E+00	0.9787E+02	0.1132E+03	2
UPONLQUE	0.3671E+02	0.0000E+00	0.0000E+00	0.0000E+00	0.3671E+02	0.3671E+02	1
----							
	NO VALUES RECORDED						
----							
	NO VALUES RECORDED						
TL-QUEUE	0.8258E+02	0.4046E+02	0.2336E+02	0.4899E+00	0.3671E+02	0.1132E+03	3
----							
	NO VALUES RECORDED						
NOT-BUSY	0.5439E+02	0.1653E+02	0.8266E+01	0.3040E+00	0.3119E+02	0.6981E+02	4
LAG-TIME	0.3042E+02	0.1464E+02	0.8450E+01	0.4812E+00	0.2078E+02	0.4726E+02	3
CAUSE-LA	0.1055E+03	0.1081E+02	0.7647E+01	0.1025E+00	0.9878E+02	0.1132E+03	2
----							
	NO VALUES RECORDED						
FREQIVAL	0.8028E+02	0.1858E+02	0.7024E+01	0.2315E+00	0.6075E+02	0.1133E+03	7

Figure 6: Example of the requested user-generated statistics output from the output of the example network.

## 7. AVAILABLE MANUALS

The first manual, 'Simulation using Saint: a user-orientated instruction manual', (Ref. 1), provides good general information about networks and SAINT. It explains all the features, such as task duration and attributes, which are needed to code a network. It is definitely the book to read in order to become familiar with networks and the SAINT terminology.

The second manual, 'The Saint User's Manual' (Ref. 2) contains all the information needed to input a network. It contains a description of all the input cards, the meaning of each field and the corresponding default values. Once the basic terminology of SAINT is understood, these input cards are self-explanatory. Also found in this manual is a list of error codes. These error codes report the incorrect specifications of a network; these codes are found at the bottom of the output file, FOR016.DAT, after the input information has been interpreted and displayed by the SAINT system. When this type of error does occur, there is no indication that the simulation has not been completed; it is only when inspecting the output file that the error code is detected.

The third manual, 'Documentation for the Saint simulation program' (Ref. 3) relates mainly to the actual SAINT simulation program code. It briefly explains each subprogram and gives the meaning of each variable used in the program.

AUTHOR'S NOTE AND ACKNOWLEDGEMENTS

The author is indebted to the staff of the Human Factors Group, particularly Dr. J. Manton for his advice and assistance during the course of this work.

REFERENCES

1. Wortman David B., Duket Steven D., Seifert Deborah J.,  
Hann Reuben L., Chubb Gerald P.,  
'Simulation using Saint : A user-orientated instruction manual.'  

---

Aerospace Medical Research Laboratory (AMRL).  
Wright-Patterson Air Force Base, OHIO, USA.  
Report Code : AMRL-TR-77-61, July 1978.
2. Wortman David B., Duket Steven D., Seifert Deborah J.,  
Hann Reuben L., Chubb Gerald P.,  
'The Saint User's Manual.'  

---

Aerospace Medical Research Laboratory (AMRL).  
Wright-Patterson Air Force Base, OHIO, USA.  
Report Code : AMRL-TR-77-62, July 1978.
3. Wortman David B., Duket Steven D., Seifert Deborah J.,  
Hann Reuben L., Chubb Gerald P.,  
'Documentation for the Saint simulation program.'  

---

Aerospace Medical Research Laboratory (AMRL).  
Wright-Patterson Air Force Base, OHIO, USA.  
Report Code : AMRL-TR-77-63, July 1978.

## APPENDIX 1 : Modifications made to the initial SAINT simulation program.

## (1) Change 1 - missing line.

2011 IF (KREAD(17).EQ.0) GO TO 3000	00011820
WRITE(NPRNT,3580)	00011830
GO TO 2970	00011840
2021 IF (KREAD(18).EQ.0) GO TO 3000	00011850
WRITE(NPRNT,3610)	00011860
GO TO 2970	00011870
C ***** The next statement was missing - it was obvious what it was	
C supposed to be : modified by Sabrina Sestito *****	
2031 if (kread(19).eq.0) go to 3000	
WRITE(NPRNT,3600)	00011890
GO TO 2970	00011900
2041 IF(KREAD(20).EQ.0) GO TO 3000	00011910
WRITE(NPRNT,3290)	00011920

## (2) Change 2 - random number generator function - VAX compatible.

END	00019960
FUNCTION DRAND(IY)	00019970
C *****	
C This function was modified by Sabrina Sestito, to make this	
C function compatible with the VAX VMS computer.	
C *****	
DRAND=РАН(IY)	
C ***** Below, is the original code for this function *****	
C IF(IY) 5,6,6	00019990
C 5 IY=IY+2147483647+1	00020000
C 6 YFL=IY	00020010
C DRAND=YFL*.4656613E-9	00020020
C	00020030
C RETURN	00020040
C	00020050
END	00020060
SUBROUTINE ENDIT(NITER)	00020070

## (3) Change 3 - making files 18 &amp; 19 temporary.

---

	IF (IFIN.EQ.0) GO TO 100	00001110
C	*****	
C	This part was inputted by Sabrina Sestito - it's purpose is	
C	to delete the temporary files NRNIT(18) and NRENT(19)	
	CLOSE(NRNIT,STATUS='DELETE')	
	CLOSE(NRENT,STATUS='DELETE')	
C	*****	
	STOP	00001120
C		00001130
	END	00001140
	SUBROUTINE DFAUS	00001150

---

## (4) Change 4 - changing the format statement.

---

	5040 FORMAT(1H1,49X,33H*INITIAL SYSTEM ATTRIBUTE VALUES*,1X//	00008380
	*55X,9HATTRIBUTE,5X,9HATTRIBUTE,1X/56X,6HNUMBER,9X,5HVALUE,1X/)	00008390
C	*****The following format statement was slightly*****	
C	*****changed from 'f8.3' to 'f10.3' - Sabrina Sestito*****	
	5050 FORMAT(100(57X,I4,8X,F10.3/))	00008400
C		00008410

---

APPENDIX 2 : The input, under the file 'FOR015.DAT', needed for the example network.

```

GEN,SABRINA,2,6,1986,1,1*
POP,(4)2,7,5*
OUT,1,1,(14)N*
ISA,1,SC,500*
UBO,1,NEWCON,2,UPCLASS,3,UPONLY*
UBO,5,NEWCONQUE,6,UPCLQUE,7,UPONLQUE*
UBO,11,TL-QUEUE*
UBO,13,NOT-BUSY,14,LAG-TIME,15,CAUSE-LAG*
UBO,17,FREQVALUE*
DIS,1,UN,,2,27*
DIS,2,UN,,3,87*
DIS,3,UN,,2,20*
DIS,4,UN,,10,32*
DIS,5,CO,5.25*
DIS,6,UN,,9,36*
DIS,7,CO,5.25*
DIS,8,UN,,5,42*
DIS,9,CO,5.25*
DIS,10,UN,,60,120*
TAS,1,BEGIN,0,1,SC,0,,,SO*
ATA,1,REL,SA,,2,DS,10,SA,,6,UF,2,SA,,3,UF,1*
PRO,1,,,2,0.171,3,0.195,4,0.634*
TAS,2,NEW-CONTACT,1,1,DS,1*
ATA,2,REL,IA,,1,DS,4,IA,,2,DS,5,SA,,7,SC,2*
DET,2,5*
TAS,3,UPDATE/CLASS,1,1,DS,2*
ATA,3,REL,IA,,1,DS,6,IA,,2,DS,7,SA,,7,SC,3*
DET,3,5*
TAS,4,UPDATE-ONLY,1,1,DS,3*
ATA,4,REL,IA,,1,DS,8,IA,,2,DS,9,SA,,7,SC,4*
DET,4,5*
TAS,5,RECEIVE,1,1,SC,0*
PRO,5,,,6,0.151,8,0.849*
TAS,6,NO-RECORD,1,1,SC,0*
PRO,6,,,7,0.5,99,0.5*
TAS,7,REPOOW,1,1,SC,1*
MOD,7,2,A,T*
DET,7,99*
TAS,8,RECORD,1,1,SC,1*
MOD,8,1,A,T*
PRO,8,,,9,0.5,99,0.5*
TAS,9,REPOOW,1,1,SC,1*
MOD,9,2,A,T*
DET,9,99*
TAS,99,CHECK,1,1,SC,1*
ATA,99,REL,SA,,4,UF,1,SA,,5,UF,2*
STA,99,,,NUM,REL*
MOD,99,5,A,T,4,A,T*
CFI,99,1,TLA,1,,SA,,100,TGA,1,,SA*
TAS,100,FINISH,1,1,SC,0,,,SI*
MOD,100,6,A,T*
FIN

```

APPENDIX 3 : The code for the SUBROUTINE MODRF and FUNCTION USERF needed for the example network.

```

SUBROUTINE MODRF(MFN,NNODE)
  COMMON /COM22/ TTIME,PFIRB
  COMMON /COM06/ TNOW,TINEX,MFAD,SEED,ISEED,NCROR,NPRNT,NPUNCH,
  * NRNIT,NRENT,MNDC,NDC,NDTN,NNTC

  IF (MFN.EQ.4) GOTO 10
  IF (MFN.EQ.5) GOTO 20
  IF (MFN.EQ.6) GOTO 30

C   THIS MODERATOR FUNCTION ASSIGNS THE TASK PERFORMANCE TIME
C   TO THE VALUE OF THE INFORMATION ATTRIBUTE, #MFN.
  CALL GETIA(MFN,VALUE)
  TTIME=VALUE
  RETURN

C   CODE FOR MODERATOR FUNCTION 4
C   THIS MODERATOR FUNCTION DETERMINES THE TASK PERFORMANCE TIME
C   OF THE 'CHECK' TASK. THIS IS DEPENDED UPON THE TIME THAT IT
C   TOOK TO TRAVERSE ONE BRANCH AND THE FREQUENCY TIME OF THE
C   MESSAGES
10  CALL GETSA(2,FREQT)
    CALL GETSA(5,BRANCHT)
    CALL GETSA(6,BRANLAST)
    CALL GETSA(7,TASKNO)
    ITASKNO=INT(TASKNO)

    CALL UCLCT(FREQT,17)

    CURLAG=FREQT-BRANCHT+CURLAG

    IF (CURLAG.GT.0) THEN
      TTIME=CURLAG
      CALL UCLCT(TTIME,13)
      CURLAG=0
    ELSE
      CALL UCLCT(-CURLAG,14)
      CALL UCLCT(BRANCHT,11)
      CALL UCLCT(BRANCHT,ITASKNO+3)
      TTIME=0
      IF (FREQT-BRANCHT.LT.0) THEN
        CALL UCLCT(BRANCHT,15)
      END IF
    END IF

C   CHECKING TO SEE WHETHER ARE AT THE END OF SIMULATION
  CALL GETSA(1,TIMELIMIT)
  IF (TNOW.GT.TIMELIMIT) TTIME=0

  RETURN

C   CODE FOR MODERATOR FUNCTION 5
C   KEEPS A RECORD OF THE USER-GENERATED VALUES

```

```

20      CALL GETSA(5,TIMETRAV)
      CALL GETSA(7,TASKNO)
      IF (TASKNO.EQ.2) CALL UCLCT(TIMETRAV,1)
C      NOTE : CALL UCLCT(XK,-1) WILL PRINT OUT EACH INDIVIDUAL OBS,
C      WITH AN ACCUMULATED RECORD OF OBS.
      IF (TASKNO.EQ.3) CALL UCLCT(TIMETRAV,2)
      IF (TASKNO.EQ.4) CALL UCLCT(TIMETRAV,3)
      IF ((TASKNO.NE.2).AND.(TASKNO.NE.3).AND.(TASKNO.NE.4)) then
          PRINT *, ' TASKNO IS INCORRECT ',TASKNO
      END IF
25      RETURN

C      CODE FOR MODERATOR FUNCTION 6
C      PRINTS OUT ALL THE USER-GENERATED STATISTICS
30      CALL UCLCT(XK,0)
      RETURN

      END

      FUNCTION USERF(JJ)
      COMMON /COM06/ TNOW,TNEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
      * NRNIT,NRENT,MNDC,NDC,NDTN,NNTC
C      USERF=1.
      GO TO (10,20), JJ

C      CODE FOR USERFUNCTION 1
10      USERF=TNOW
      RETURN

C      CODE FOR USER FUNCTION 2
20      CALL GETSA(3,VALUE3)
      CALL GETSA(4,VALUE4)
      USERF=VALUE4-VALUE3
      RETURN

      END

```

---

## DISTRIBUTION

### AUSTRALIA

#### Department of Defence

##### Defence Central

Chief Defence Scientist  
Deputy Chief Defence Scientist (Shared Copy)  
Superintendent, Science and Program Administration (Shared Copy)  
Controller, External Relations, Projects and  
Analytical Studies (Shared Copy)  
Director, Departmental Publications  
Counsellor, Defence Science (London) (Doc Data Sheet Only)  
Counsellor, Defence Science (Washington) (Doc Data Sheet Only)  
S.A. to Thailand MRD (Doc Data Sheet Only)  
S.A. to the DRC (Kuala Lumpur) Doc Data Sheet Only )  
OIC TRS, Defence Central Library  
Document Exchange Centre, DISB (18 copies)  
Joint Intelligence Organisation  
Librarian H Block, Victoria Barracks, Melbourne  
Director General - Army Development (NSO) (4 copies)  
Defence Industry and Materiel Policy, FAS

##### Aeronautical Research Laboratories

Director  
Library  
Authors: S. Sestito  
P.F. Preston  
D.H. Spivakovsky  
J.G. Manton  
B.A. Clark  
A. Ong  
R.L. Nathan

##### Materials Research Laboratories

Director/Library

##### Defence Research Centre

Library

##### RAN Research Laboratory

Library

##### Navy Office

Navy Scientific Adviser  
Aircraft Maintenance and Flight Trials Unit  
RAN Technical School, Library  
Director of Naval Aircraft Engineering  
Director of Naval Air Warfare  
Superintendent, Aircraft Maintenance and Repair  
Director of Naval Ship Design  
Director of Operational Analysis - Navy

**OIC RANTAU**

**Director of Tactics, Action, Information, Organization and Navigation**

**Director of Psychology - Navy**

**OC HMAS Platypus**

**Army Office**

**Scientific Adviser - Army (Doc Data sheet only)**

**Engineering Development Establishment, Library**

**US Army Research, Development and Standardisation Group**

**Director of Operational Analysis - Army**

**Director of Psychology - Army**

**Air Force Office**

**Air Force Scientific Adviser (Doc Data sheet only)**

**Director of Operational Analysis - Air Force**

**Director of Psychology - Air Force**

**Central Studies Establishment**

**Information Centre**

**Department of Aviation**

**Library**

**Flight Standards Division**

**Statutory and State Authorities and Industry**

**Aerospace Technologies Australia, Manager/Library**

**Universities and Colleges**

**Adelaide**

**Barr Smith Library**

**Flinders**

**Library**

**La Trobe**

**Library**

**Melbourne**

**Library**

**Monash**

**Hargrave Library**

**Newcastle**

**Department of Electrical and Computer Engineering**

**New England**

**Library**

**Sydney**

**Engineering Library**

**NSW**

Physical Sciences Library  
Library, Australian Defence Force Academy

**Queensland**  
Library

**Tasmania**  
Engineering Library

**Western Australia**  
Library

**RMIT**  
Library

**UNITED STATES OF AMERICA**

Armstrong Aerospace Medical Research Laboratory  
Defence and Civil Institute of Environmental Medicine

SPARES (10 copies)  
TOTAL (89 copies)

AL 140  
REVISED APRIL 87

DEPARTMENT OF DEFENCE

# DOCUMENT CONTROL DATA

PAGE CLASSIFICATION  
**UNCLASSIFIED**

PRECEDENCE MARKING

10. AS NUMBER <b>AR-004-554</b>	11. ESTABLISHMENT NUMBER <b>ARL-SYS-TM-93</b>	2. DOCUMENT DATE <b>7 SEPTEMBER 1987</b>	3. TASK NUMBER <b>DST 85/037</b>
4. TITLE <b>A WORKED EXAMPLE OF AN APPLICATION OF THE SAINT SIMULATION PROGRAM</b>		5. SECURITY CLASSIFICATION <small>PLACE APPROPRIATE CLASSIFICATION BY ORK OR SE. SECRET OR. CONFIDENTIAL OR. RESTRICTED OR. UNCLASSIFIED OR.</small> <div> <div>U</div> <div>U</div> <div>U</div> </div> <div>DOCUMENT TITLE ABSTRACT</div>	6. No. PAGES <b>29</b>
8. AUTHOR <b>S. SESTITO</b>		9. COMMISSIONING/DELIVERY INSTRUCTIONS <b>NOT APPLICABLE</b>	
10. CORPORATE AUTHOR AND ADDRESS  <b>AERONAUTICAL RESEARCH LABORATORIES P.O. BOX 4331, MELBOURNE VIC. 3001</b>		11. OFFICE/POSITION RESPONSIBLE FOR SPONSOR _____ SECURITY _____ COORDINATION _____ APPROVAL _____	
12. SECURITY RESTRICTIONS OF THIS DOCUMENT <b>Approved for public release</b>			
<small>OVERSEAS DISSEMINATION OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH AUSA. DEFENCE INFORMATION SERVICES BRANCH, DEPARTMENT OF DEFENCE, CAMPBELL PARK, CANBERRA, ACT 2601</small>			
13a. THIS DOCUMENT MAY BE ADVERTISED IN CATALOGUES AND ADVERTISING SERVICES AVAILABLE TO <b>No limitations</b>			
13b. CITATION FOR OTHER PURPOSES (IE. CABLE ASSIGNMENT) MAY BE <input checked="" type="checkbox"/> UNRESTRICTED OR <input type="checkbox"/> AS PER 13a.			
14. DESCRIPTION <b>SAINT (Systems Analysis of Integrated Networks of Tasks): SAINT programming language, Computerized Simulation, Network modelling, Man machine systems, Operator Performance, Human Factors Engineering.</b>			15. ORCA SUBJECT CATEGORIES  <b>0062B 0095D</b>
16. ABSTRACT <b>SAINT is a network modelling and simulation technique developed to assist in the design and analysis of complex human-machine systems. This document discusses some of the SAINT concepts, the development of a SAINT network, the method of inputting the network into the SAINT environment, and presents a brief look at the output from SAINT.</b>			

PAGE CLASSIFICATION

UNCLASSIFIED

PREVAILING MARKING

THIS PAGE IS TO BE USED TO RECORD INFORMATION WHICH IS REQUIRED BY THE ESTABLISHMENT FOR ITS OWN USE BUT WHICH WILL NOT BE ADDED TO THE DISTIS DATA UNLESS SPECIFICALLY REQUESTED.

16. ABSTRACT (CONT.)

17. DISPROFIT

AERONAUTICAL RESEARCH LABORATORIES, MELBOURNE

18. DOCUMENT SERIES AND NUMBER

SYSTEMS TECHNICAL  
MEMORANDUM 93

19. COST CODE

73 4676

20. TYPE OF REPORT AND PERIOD COVERED

21. COMPUTER PROGRAMS USED

SAINT

22. ESTABLISHMENT FILE REF. NO.

23. ADDITIONAL INFORMATION (AS REQUESTED)

END

DATE  
FILMED

2 88